IES de Teis
Avda. de Galicia, 101
36216 – Vigo

Tfno: 886 12 04 64
e-mail: ies.teis@edu.xunta.es
http://www.iesteis.es

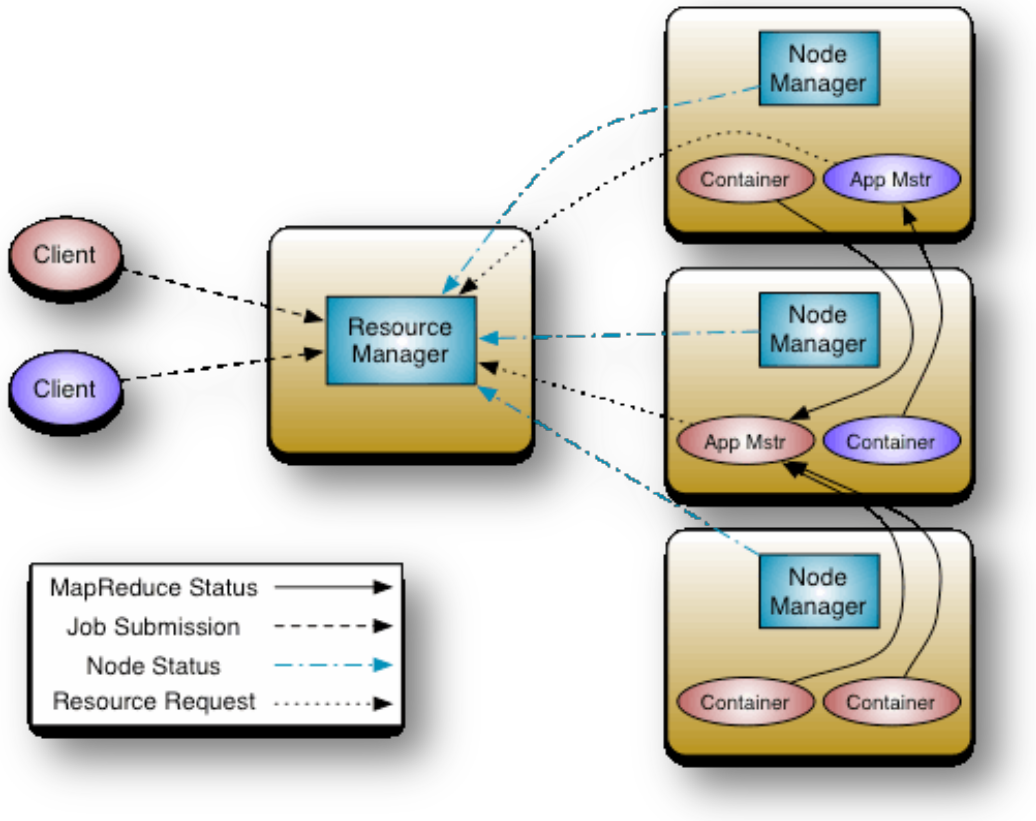
Unidad didáctica 3. YARN

En las primeras versiones de Hadoop existía, igual que ahora, la capa de almacenamiento HDFS y se utilizaba MapReduce para el procesamiento distribuido. El problema es que la gestión de los recursos de los nodos era complicada y dependía del programador.

En la actualidad, Hadoop incluye una nueva capa justo por encima de HDFS que se encarga de administrar y asignar los recursos de los nodos para el procesamiento de datos. Esta capa se llama YARN y viene de "Yet Another Resource Negotiator".

La asignación de recursos que hace YARN es mediante lo que se llama contenedores. Un contenedor es un bloque compuesto por tiempo de procesador y memoria en un determinado nodo.

YARN tiene una estructura similar a HDFS en la que también hay un nodo principal llamado "Resource Manager" por la función que desempeña, y múltiples nodos trabajadores llamados "Node Manager" también por la función que desempeñan. Ambos constituyen el framework del procesamiento distribuido.



Contenedores

Ya sabemos que los programas se alojan en disco pero se ejecutan en procesos de memoria principal. El tiempo de procesador dedicado a cada uno de ellos es algo que depende del sistema operativo y, aunque podemos influir ligeramente alterando las prioridades del proceso, lo cierto es que la última decisión la toma el sistema operativo. En cambio reservar un espacio de memoria sí es algo que se puede hacer directamente a través del sistema operativo.

Por ello, en Hadoop los contenedores son principalmente reservas de memoria para la ejecución de trabajos. Además pueden incluir otros componentes como el sistema de ficheros distribuidos HDFS. Los contenedores permiten empaquetar todos los servicios y configuraciones necesarios en un entorno aislado y portátil. Estas características permiten que un bloque su pueda portar a otro nodo en caso necesario.

Los contenedores nacen en base a las peticiones que hacen las aplicaciones y a la configuración del sistema. Por ejemplo, en un clúster con 100 gigas de memoria y con un tamaño mínimo de contenedor de

1 giga podríamos tener 100 contenedores. Si el tamaño mínimo del contenedor es de 4 gigas podríamos tener 25 contenedores. Cuando un trabajo MapReduce solicita memoria se le asigna el número mínimo necesario de contenedores. Por ejemplo, si en este último caso la aplicación solicita 5 gigas tendremos que asignarle dos contenedores de 4 gigas.

En la configuración de YARN tenemos varias propiedades referentes a los contenedores que podemos personalizar como por ejemplo:

- Yarn.scheduler.minimun-allocation-mb
- Yarn.scheduler.maximum-allocation-mb
- Yarn.nodemanager.vmem-pmem-ratio
- Yarn.nodemanager.resource.memory.mb
- Mapreduce.map.memory.mb
- Mapreduce.reduce.memory.mb

Por el nombre de la propiedad ya podemos hacernos una idea de qué es lo que estamos configurando. Por ejemplo, si en nuestro clúster la mayoría de los "mappers" son ligeros y lo que queremos es que se pueda procesar muchos trabajos, es probable que una configuración a la baja sea las más apropiada. Para pocos trabajos pero muy pesados probablemente una configuración asignando mucha memoria sea la más adecuada. También será necesario tener en cuenta que si nuestros nodos aportan 4 gigas de memoria cada uno, difícilmente podremos alojar un contendor de 8 gigas.

Application Master

El "Application Master" en un contenedor especial que se despliega en primer lugar en alguno de los nodos del clúster cuando un nuevo trabajo MapReduce entra en la capa YARN. Su misión es gestionar y coordinar los contendores necesarios para ejecutar esa aplicación durante todo su tiempo de vida. Esos otros contenedores pueden estar en ese o en cualquier otro nodo.

Al cargar el trabajo, el "Application Master" solicitará al nodo principal un determinado número de contenedores para satisfacer las necesidades del trabajo. Además también coordina la ejecución de las tareas de MapReduce. Otra de las funciones que tiene es monitorizar el estado de los contenedores para mostrar esa información así como solucionar los fallos reiniciando el contenedor o solicitando otro distinto.

Cada trabajo tendrá su propio "Application Master". Si nos fijamos en el gráfico inicial veremos dos clientes que despliegan dos aplicaciones distribuidas. Cada una de estas aplicaciones tiene un

"Application Master" en alguno de los nodos, en este caso se distinguen bien porque tienen el mismo color que su cliente.

Node Manager

Este componente se ejecuta en cada nodo del clúster YARN por eso le da nombre al tipo de nodo. Su misión es la de administrar los recursos hardware locales en ese nodo y coordinar la ejecución de los contenedores que existan en ese nodo.

Otra de sus responsabilidades es la creación, puesta en marcha, monitorización y gestión de los contenedores de ese nodo. El "Node Manager" administra parcialmente el tiempo de ejecución asignado a cada contendor ya creado estableciendo colas o sistemas de prioridades.

El "Node Manager" mantendrá informado regularmente al nodo principal sobre el estado del nodo incluyendo información sobre los recursos disponibles y la carga actual.

Resource Manager.

La capa YARN mantiene la misma filosofía de lo que ya henos visto en Hadoop, una arquitectura distribuida centralizada. Esto quiere decir que hay un nodo con un papel principal mientras que el resto son nodos trabajadores. Este sistema tiene un punto de fallo claro, el nodo principal pero a cambio es rápido y sencillo de implementar. En la capa YARN, ese nodo principal se llama "Resource Manager" como el nombre servido que se ejecuta en él.

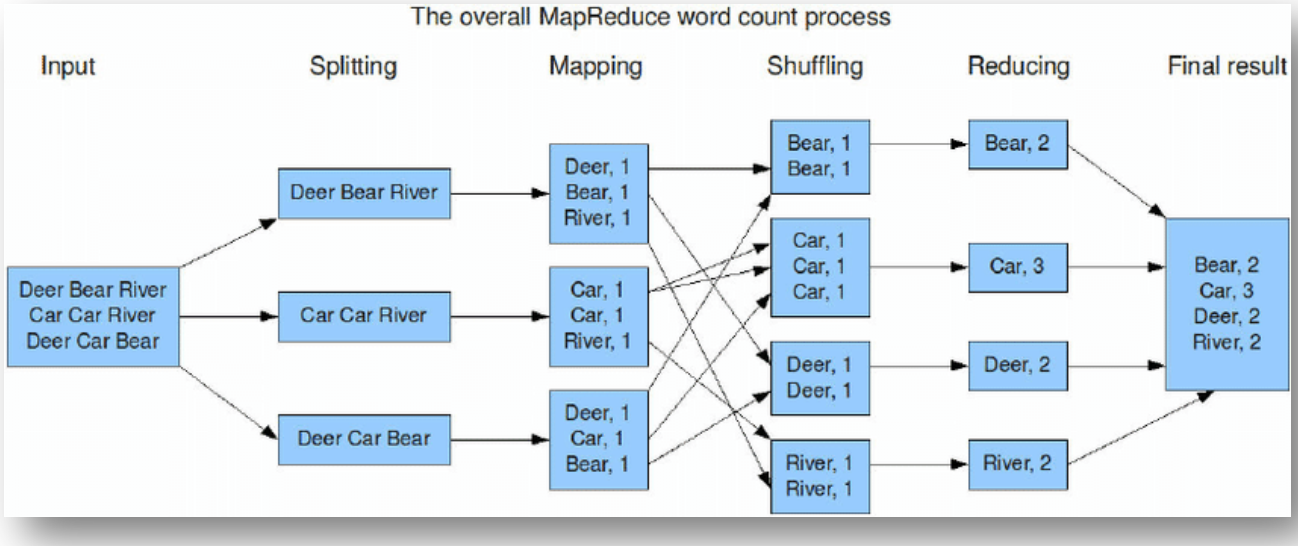
En realidad el "Resource Manager" tiene dos componentes principales, el "Scheduler" o planificador y el "Applications Manager".

El "Scheduler" o planificador es el encargado de reservar contenedores para las aplicaciones en ejecución. Una de las propiedades que podemos modificar en la configuración de YARN es el tipo de planificador que se va a usar. Podemos elegir entre algunos como "CapacityScheduler" o "FairScheduler" que se diferencian en las estrategias usadas para distribuir los recursos, colas con prioridades en el primer caso y colas con rotación en el segundo caso.

El "Applications Manager" es el responsable de aceptar nuevos trabajos, negociar el primer contenedor para el "Application Master" y monitorizarlo por si es necesario reiniciarlo. Además el "Application Master" debe negociar con el planificador el resto de los contenedores y llevar cuenta de su estado y progreso.

Funcionamiento de la capa YARN

Ahora que ya conocemos todos los actores que intervienen podemos explicar y entender el funcionamiento de esta capa YARN de procesamiento distribuido.



1.- Un cliente establece comunicación con el "Resource Manager" y solicita ejecutar un trabajo.

2.- El "Resource Manager" valora el estado y capacidad de los nodos para aceptar o no el trabajo. Si no lo rechaza negociará con el planificador los nodos y recursos necesarios para ejecutar la tarea MapReduce.

3.- El "Resource Manager" seleccionará un nodo y lanzará allí el primer contendor de ese trabajo donde se ejecutará un "Application Master" propio para ese trabajo

4.- Será el "Node Manager" el que ejecute esta tarea de creación y, una vez que el "Application Master" ha registrado su presencia en el "Resource Manager" recibirá la información sobre los recursos que finalmente se le han asignado.

5.- Si fuera necesario, el "Application Master" solicita contenedores adicionales al planificador del "Resource Manager".

6.- Los contenedores asignados para las tareas "map" y "reduce" son lanzadas por los "Node Manager".

7.- El "Node Manager" informa regularmente al "Resource Manager" del estado del nodo.

8.- El "Application Master" informa regularmente al "Resource Manager" del progreso del trabajo.

9.- Una vez finalizado el trabajo, el "Application Master" informa al "Resource Manager", se detiene y libera los contenedores usados.

MapReduce

Hablar de MapReduce es hablar de un modelo de programación orientado al procesamiento de datos distribuidos donde se tratan grandes conjuntos de datos en un clúster de múltiples nodos.

De manera muy resumida, el modelo MapReduce se basa en dos funciones principales, "Map" y "Reducer". La función "Map" toma como entrada un conjunto de datos en bruto y los procesa hasta un estado intermedio en forma de pares de clave-valor. La función "Reduce" parte de esos pares y los procesa para generar los resultados finales.

Mappers

Los "mappers" tienen como misión procesar un fragmento de los datos. Estos datos se dividen en registros como una base de datos o líneas de texto y cada uno de ellos se procesa individualmente.

Este proceso consiste en producir un conjunto de pares clave-valor. La clave identifica la información relevante del registro y el valor contiene los datos asociados a esa llave.

Los pares clave-valor son enviados a un sistema de ordenación y agrupación donde se combinan con otros pares clave-valor con la misma clave.

Las funciones "Mapper" se ejecutan de forma paralela en el mismo nodo y distribuida en distintos nodos lo que permite un escalado horizontal con el que manejar grandes volúmenes de datos.

Reducers

Un "reducer" es una función que toma como entrada la lista de pares clave-valor combinadas de los "mappers". Para simplificar la implementación de los "reducers", los datos de entrada ya están ordenados y agrupados por clave. De esta manera podemos asegurar que los valores asociados con una misma clave se encuentran en el mismo "reducer".

La función "reducer" puede realizar operaciones sobre estos datos, especialmente con la parte de valor en la que puede realizar operaciones de agregación como sumar valores o calcular estadísticas. También puede filtrar datos o realizar alguna otra operación personalizada.

El resultado de procesamiento es también uno o varios pares clave valor finales que representan el procesamiento de esa clave en particular. El proceso itera con todas las claves y finalmente escribe el resultado en el sistema de ficheros como HDFS o una base de datos.

Shuffle

Entre la etapa de los "mappers" y los "reducers" hay una fase de ordenación y mezcla llamada simplemente "Shuffle"

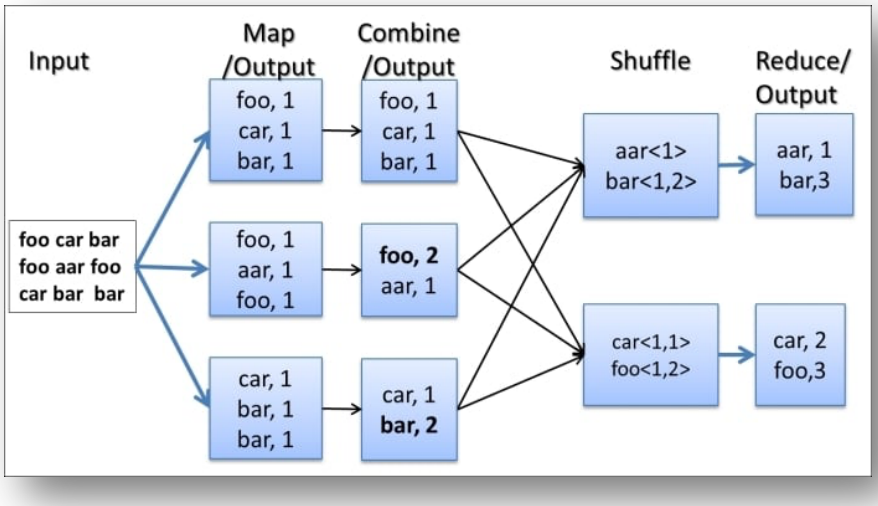
En la fase de ordenación los pares de un "mapper" se ordenan en función de su clave. Esta fase supone una inversión futura porque simplifica enormemente la gestión de los registros porque pasar de una clave a otra implica que la primera ya no vuelve a aparecer en los datos.

La siguiente etapa es la de "Shuffle" en la que se unen todas las claves iguales de los distintos "mappers". Además también se forman bloques en los que se da prioridad a que todas las claves iguales formen parte del mismo bloque. Al estar los datos ordenados realizar esta partición comparando resultados de distintos "mappers" se simplifica mucho.

Imaginemos que las claves son las letras de abecedario. No necesariamente en todos los "mappers" vamos a tener una clave "A", lo que si podemos asegurar es que después de la etapa de ordenación, todas las claves "A" estarán en las primeras líneas de los nodos que sí las tuvieran. Los nodos que no tienen "A" en la primera posición ya quedan descartados, para los que tengan "A" se irán recorriendo hasta que encuentren otro valor en la clave. Todos los pares en los que figure la clave "A" formarán parte de un mismo bloque. El proceso se repite hasta completar el bloque, momento en el que se tendrá que abrir otro bloque.

Combiners

Opcionalmente podemos incluir una etapa intermedia entre los "mappers" y la etapa "Shuffle". En general los "mappers" procesan los datos de una manera muy atómica, de elemento en elemento, provocando que muchos pares se repitan. Un "combiner" es una función que hace una agregación previa al ordenado de manera que las escrituras que se hacen a disco durante la etapa "Shuffle" se reduzcan mucho ganando con ello un aumento de la velocidad importante.



Ajuste de las propiedades YARN

En los gráficos que acompañan este apartado podemos seguir las fases de un trabajo MapReduce y situar donde intervienen algunas de las propiedades que podemos personalizar en Hadoop.

Algunas como "dfs.blocksize" ya las hemos nombrado en temas anteriores. Se trata del tamaño del bloque en el que HDFS trocea los archivos y con los que hace las réplicas. Sabemos que por defecto tiene un tamaño de 128 megabytes pero que podemos adaptarlo a nuestras necesidades. Lo más eficiente es que cada bloque se asocie a un "mapper" y ambos residan en el mismo nodo. Si tenemos un dataset pequeño podemos procesarlo en un solo nodo pero, si por alguna razón insistimos en que se procese de manera distribuida sería bueno configurar el tamaño del bloque mucho más pequeño.

También es posible modificar algunos parámetros relacionados con la etapa de ordenación dentro la fase "Shuffle". Solo llamaré tu atención sobre la propiedad "mapreduce.job.local.dir" en la que podemos especificar la ruta HDFS donde se guardarán los archivos temporales de la ordenación. Es interesante porque deja claro que en esta fase se escribe en disco. Ya sabemos que escribir en disco penaliza mucho en términos de velocidad, escribir en HDFS aun penaliza más. En el próximo tema, cuando veamos Spark, explicaremos que es otro motor de procesamiento distribuido mucho más rápido porque trabaja en memoria y evita tener que hacer escrituras en disco. Ya lo veremos.

En la parte final de la etapa "Shuffle" también se pueden modificar propiedades que tienen que ver con el uso de la memoria e hilos de ejecución usados para crear los bloques que servirán de entrada para la etapa "reducer". Es una manera algo complicada de manipular cuantos "reducers" queremos usar.

Maximizar todo lo posible la memoria creará bloques grandes y, por tanto menos "reducers". Minimizar el uso de memoria creará muchos bloques con menos contenidos que necesitarán muchos "reducers".

Como siempre, el ajuste fino que realizará el administrador depende mucho del problema y de cómo se pretenda afrontarlo